International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 5 Issue 4, May-June 2021 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

A Traffic Sign Classifier Model using Sage Maker

Arpit Seth[1], Vijayakumar A[2]

[1]Student, [2]Professor,
[1,2]Department of MCA School of CS & IT, Jain Deemed-to-be University, Bengaluru, Karnataka, India

ABSTRACT

Driver assistance technologies that relieve the driver's task, as well as intelligent autonomous vehicles, rely on traffic sign recognition. Normally the classification of traffic signs is a critical challenge for self-driving cars. For the classification of traffic sign images, a Deep Network known as LeNet will be used in this study. There are forty-three different categories of images in the dataset. There are two aspects to this structure: Traffic sign identification and Traffic sign classification. ADASs are designed to perform a variety of tasks, including communications, detection of road markings, recognition of road signs, and detection of pedestrians. There are two aspects to this structure: Traffic sign identification and Traffic sign classification. In the methodologies for detecting and recognizing traffic signals various techniques, such as colour segmentation and the RGB to HSI model area unit, were applied for traffic sign detection and recognition. Different elements contribute to recognition of HOG.

How to cite this paper: Arpit Seth | Vijayakumar A "A Traffic Sign Classifier Model using Sage Maker" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-5 | Issue-4, June 2021, pp.855-858, URL: www.ijtsrd.com/papers/ijtsrd42411.pdf

IJTSRD42411

I. INTRODUCTION

Traffic sign recognition has a lot of economic promise in clever autonomous vehicles and driver assistance systems. Without correctly deploying and maintaining road traffic signs, traffic signals, and road markings, it is impossible to improve traffic quality and safety. Amazon Sage Maker could be a machine-learning platform in the cloud that allows developers to create, test, and deploy machine-learning models.

We frequently use Amazon Sage Maker to create models. Sage Maker has a number of advantages, including a large amount of processing capacity (various CPUs and GPUs with parallel computing) and hence the possibility to deploy the model as a whole. The entire analysis was completed in Sage Maker's notebook, while training and prediction were handled by a large number of strong GPU instances. This paragraph can encompass the entire pipeline, from pre-processing to final model, for the sake of completeness. A variety of classification techniques have been used to recognize traffic signals, ranging from simple model matching (e.g. pooling, flattening) to complex Machine learning techniques (e.g. support vector machines, boosting, dense neural network etc.). Furthermore, moving from isolated frames to feature maps for traffic sign analysis can help to reduce the number of false alarms while also improving the precision and accuracy of the detection and recognition process.

The training dataset, testing dataset, and validation dataset are the three elements of the dataset in this work. Accidents can occur for a variety of causes, including drivers failing to notice a proof in a timely manner or neglecting to listen at critical times. In unfavourable weather circumstances such as severe rain showers, fog, or falls, drivers pay little attention to traffic signals and focus instead on driving. Control, in addition to guiding and advising drivers, necessitates automatic traffic sign recognition. In general, traffic signs provide the driving force with critical information for safe and efficient navigation. In order to better parse and recognize the highway, self-driving cars need traffic sign recognition. Similarly, in order to assist and protect drivers, "driver alert" systems in cars must consider the roadway around them. Computer vision and deep learning can help with a variety of issues, including traffic sign recognition. Our goal is to build a multi classifier model based on deep learning to classify various traffic signs.

II. LITERATURE REVIEW

The three most popular types of traffic sign detection systems are color-based, shape-based, and learning-based. Meanwhile, Liu introduces two new modes: color and shape-based methods and LIDAR-based methods. Since it achieves a detection rate of more than 90%, the learning-based detection method is the most powerful of these approaches. On the other hand, the most recent algorithms are still a long way from being a real-time classification system.Some researchers have obtained good results by using the HSI color space instead of RGB. For traffic sign detection, a new color space called Eigen color was proposed based on Karhunen-Loeve (KL).[1]This paper proposes the "German Traffic Sign Recognition Benchmark" dataset and competition, as well as their design and analysis. The results of the competition show that cutting-edge machine learning algorithms excel at the challenging task of traffic sign recognition.[2] The baseline algorithms considered in this

paper are the Viola-Jones detector based on Haar features and a linear classifier based on HOG descriptors, which are two of the most common detection approaches.[3] This paper focuses on real-time traffic sign recognition, which involves deciding in a short period of time which type of traffic sign appears in which region of an input image. To accomplish this, a two-module architecture (detection and classification modules) is proposed. The color probability model in the detection module converts the input color image into probability maps. The road sign recommendations are then derived by identifying the most secure extreme areas on the charts.[4] In this research, the HIS color model is used to detect the traffic sign, which is accompanied by circle detection. [5]The precise sign area cannot be determined from the color-detected regions. This approach traces the edge of interested regions to obtain their contours after morphologic operations.

III. PROPOSED METHODOLOGY

In this purposed system we use the LeNet CNN architecture to develop a multi classifier model based on deep learning to categorize multiple traffic indicators in this purposed system. Image classifiers work by guessing the types of objects in a photo. We employ a convolutional neural network as a classification model (CNN). In general, when it comes to classifying images on large datasets, CNN beats classic neural networks and other machine learning models. For the classification of traffic sign photos, a Deep Network known as LeNet will be used. Below are the steps we followed to option our goals and to achieve the final Le Net multi classifier model for traffic sign prediction

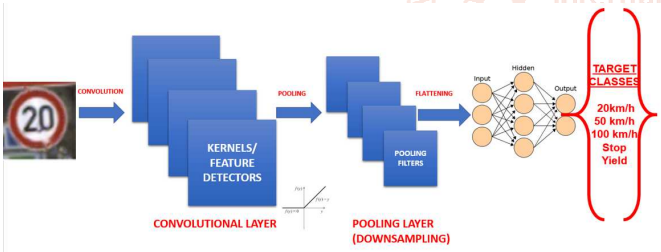

Fig 3.1 - LeNet Architecture

We collect many of these neurons together in a systematic format, in a layered fashion. After the processing we have bunch of neurons here in the hidden layer and then will connect all the neurons in one layer and will be fully connected to all the neurons in the subsequent layer this is called dense artificial neural network or fully connected artificial neural network. So let's assume that we have a bunch of images here. And these are the images from the traffic sign data set. Here we have all the different classes. A dense, fully connected artificial neural network is created. However, in order to interact with photos, we must first do extra processing procedures. Following that, we perform convolution. Then we'll employ non-linearity. Using the RELU activation function, or rectified linear unit activation function, and subsequently pooling, we hope to add non-linearity to the CNN after the convolution layer. In the pooling layer, we aim to compress all of these feature maps, all of the outputs that we extract or collect from the first layer. Pooling layers produce same findings, but in a compressed form of that information. We notice that when we use the pooling function, the size here shrinks. We simply apply flattening after we've created these pooling filters. As a result, we bring all of the pixels together. We flatten them up. After final step model will be ready to take all these inputs and feed them to dense or fully connected artificial networks.

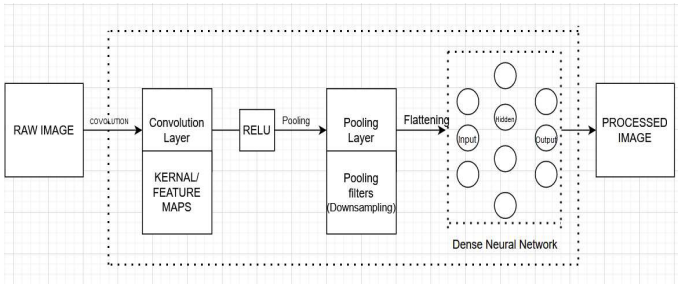

Fig 3.2- Proposed FunctionalArchitecture

Below are the process done to train the model.

1. Convolution Layer

First step for identification is convolutional layer. These convolutional are Kernels or feature detectors. These kernels or feature detectors just filters the image to extract features out of the original image. These kernels with the help of some 2X2 matrix filters the image by performing some algorithm functions on original image. Convolutions simply use the kernel matrix to scan a given image and apply a filter to obtain a certain effect. Where you take an image, you apply kernel or a feature detector to extract features out of my image. So what we do is after we apply these kernels or feature detectors, we actually generate feature maps. Feature maps are kind of different variations of the original image, as we are getting the image and we can perform sharpening to the image or blurring to the image. We will creating all these different feature maps to extract features out of original image. With the help of image kernel we have here is a matrix that is used to apply effects such as blurring and sharpening. And kernels are used in machine learning for feature extraction to select the most important pixels in the image.

2. RELU

The function map incorporates non-linearity using RELU Layers. It also improves the sparseness, or dispersion, of the function map. The RELU's gradient does not vanish as x grows, unlike the sigmoid function.

3. Pooling

Pooling or down sampling layers are placed after convolutional layers to reduce feature map dimensionality. This improves computing efficiency while keeping features intact. Pooling helps the model generalize by preventing over fitting. If one of the pixels is moved, the pooled function map will remain unaltered. Max pooling in a feature map maintains the maximum feature answer within a given sample size. The max pooling layer reduces the size of the previous layer's performance, making training easier. To avoid over fitting, a drop out layer is also used.

The sagemaker.tensorflow.TensorFlowestimator handles locating the script mode container, uploading script to a S3 location and creating a Sage Maker training job. File is created and saved in S3 bucket in .npz files so the model could access the data.

```
# Upload the training and validation data to S3 bucket

prefix = 'traffic-sign'

training_input_path   = sagemaker_session.upload_data('data/training.npz', key_prefix = prefix + '/training')
validation_input_path = sagemaker_session.upload_data('data/validation.npz', key_prefix = prefix + '/validation')

print(training_input_path)
print(validation_input_path)
```

```
s3://sagemaker-us-east-2-575195884898/traffic-sign/training/training.npz
s3://sagemaker-us-east-2-575195884898/traffic-sign/validation/validation.npz
```

Fig 3.3- upload data to S3 bucket

The training script is a python file that can be run on its own. Sage Maker will run this script on a given instance during the training job.

A. CORPORATE IMPLEMENTATION OF MACHINE LEARNING AND CLOUD COMPUTING

In every industry, machine learning and cloud computing are required for data storage, processing, and management. We offer AWS data storage and administration services as part of our cloud computing services. The data in the CSV file will be saved in the same register. The AWS files are used to collect and update data for a model that we utilize frequently. The AWS EC2 and S3 architecture is shown here, which allows us to store data and map it to the IAM service model. AWS Sage maker assists in the training of a prediction model by using built-in algorithms. You can start using AWS sage creation right immediately. Boto 3 and Sage Maker are required to be installed in the notebook. The Amazon Web Services (AWS) Software Development Kit (Boto3) is a Python version of the Amazon Web Services (AWS) Software Development Kit (SDK). Python programmers may use Boto3 to construct applications that use Amazon S3 and Amazon EC2. Then it's time to schedule a Sage maker session. Sage Maker facilitates the storing of data and training datasets in S3, as well as the training of models on S3 for later use. Sage Maker has the advantage of being able to train an unlimited number of models from which the best or most dependable model can be chosen. The Linear Learner is a supervised learning algorithm for training data by fitting a line to it.

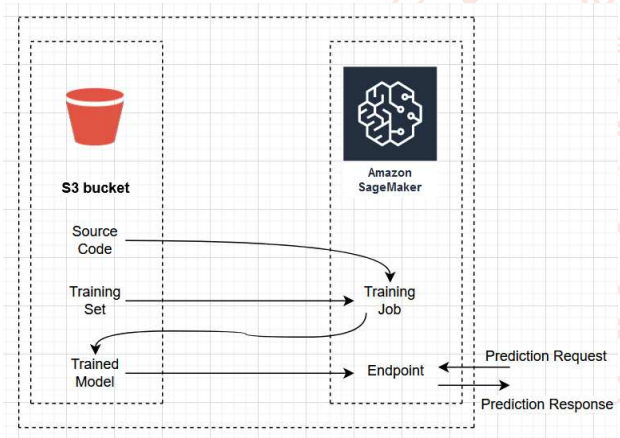

Fig 3.4 – Architectural working of sage maker with S3 Bucket

IV. RESULT and DISCUSSIONS

This is the expected outcome, and this section will explain the results as well as the example codes. In this design, we employed a multi classifier model that could accurately predict the class of pre-processed images existing in a given image. We can describe and train the model's training job.

An instance of ml.p2xlarge was used. After two epochs, the validation accuracy is 0.8523. The assignment took the instance 76 seconds to complete, with a 0.923 accuracy on the testing set.

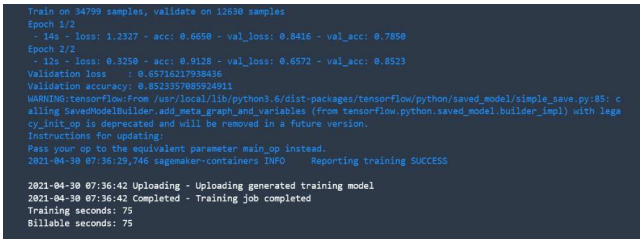

Fig 4.1- Trained Models

A. DATASET DESCRIPTION

GTSRB dataset to train our own custom traffic sign classifier. The Description of the data:

- The sizes of each picture are slightly different.
- There are 39209 training examples in total.
- The total number of test examples is 12630.
- There are 43 classes total.

The images range in scale from 20x20 to 70x70 pixels and have three channels: RGB. AWS S3 storage will be used to store all of the data. And all of the research and training is done in Sage Maker instances on the dataset. The first thing we have to do is to resize all the images to 32x32x3 and read them into a numpy array as training features.

All of the classes go through the same procedure. Data is stored in pickle form to avoid having to do pre-processing again if the notebook was disconnected. We can easily load the processed data from pickle the next time we need it.

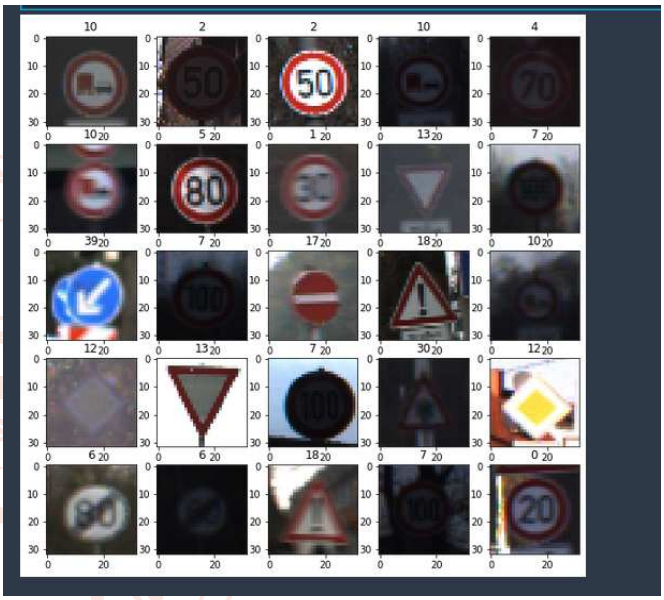

Fig 4.2 - Input Dataset

B. Final Output

The below is the sign recognition of 5 samplesfrom the testing dataset out of 43 classes.

Fig 4.3 – Predicted output from testing dataset

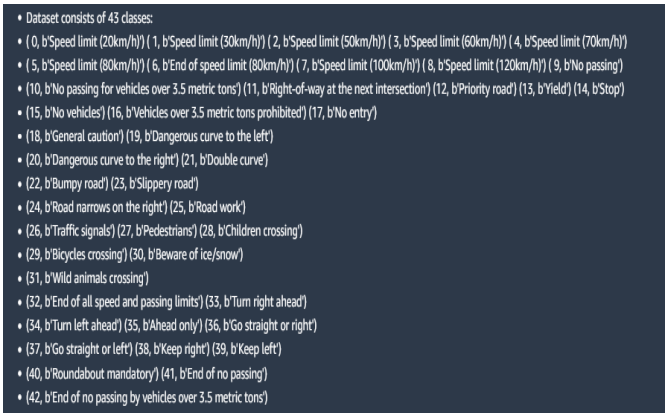

Fig 4.4 – Dataset classes

V. CONCLUTION

We used the new Tensor Flow framework to train a model for traffic sign classification. The pipeline includes pre-processing, model construction, training, estimate, and endpoint deployment. The accuracy of testing is 0.923, while the accuracy of validation is 0.8523. The accuracy of the bulk of the classes is better than 90%.

The proposed model, according to the findings, is capable of resolving some faults.

A. FUTURE WORK

The following are some suggestions for improvement:

1. This project lacks a user interface.
2. Other variables may be used to improve prediction accuracy.

References

[1] M. S. J. S. a. I. J. Stallkamp, "The German traffic sign recognition benchmark: a multi-class classification competition," *IEEE,* 2011.

[2] J. S. J. S. S. a. C. I. S. Houben, "Detection of traffic signs in real-world images: The German traffic sign detection benchmark," *IEEE,* pp. 1-8, 2013.

[3] H. L. H. X. a. F. W. Yi Yang, "Towards Real-Time Traffic Sign Detection and Classification," *IEEE,* 2014.

[4] W. L. Kai Li, "Traffic indication symbols recognition with shape contex," *IEEE,* 2011.

[5] J. K. K. J. S. Bao Trung Nguyen, "Fast traffic sign Detection under challenging conditions," *IEEE,* 2014.